

Pseudo Random Generation of the Initial Population in Genetic Algorithms: an Experiment

Pedro A. Diaz-Gomez and Dean F. Hougen
Robotics, Evolution, Adaptation, and Learning Laboratory (REAL Lab),
School of Computer Science, University of Oklahoma, OK., USA.
pdiazg@ou.edu hougen@ou.edu

1 Abstract

Genetic Algorithms (GAs) have been extensively used as heuristic tools for doing search in huge search spaces [6]. GAs take their inspiration from biology [6]: species evolved according with the environment, the ones that adapt to it, are the ones that are going to survive. The fittest are the ones that are going to reproduce and possible survive if there are enough resources [1]. In GAs a spicity is a set of individuals that usually are generated initially randomly. This set called the *initial population*, which iterates changing according to the genetic operators and a mathematical function called the fitness function [7]. So, the first step in a GA is the random generation of the first population of individuals. How does that random generation affects the performance—in term of number of iteration to find the approximate solution—of the algorithm? This is the topic of this abstract, and for doing that we are going to use as an example the finding of a longest snake in a 4-dimensional hypercube.

The Snake in the Box Problem. Longest snakes in hypercubes are of interest in coding theory [8], digital design, and telecommunications [5]. A *d-dimensional hypercube* is a connected, non-directed graph of 2^d vertexes or nodes, where each vertex has d neighbors and a binary labeling of each node may be given that differs in exactly one bit with that of each of its neighbors [9].

A *snake* is a complete, connected, open path in the d-hypercube where each node belonging to the path has two neighbors, except the head and the tail which have only one neighbor each. That is, a node in the snake is adjacent to at most two nodes in the path and there must be exactly two distinguished nodes, the head and the tail, each with only one neighbor in the path [2]. Figure 1 shows a path that is a snake in a 4-dimensional hypercube.

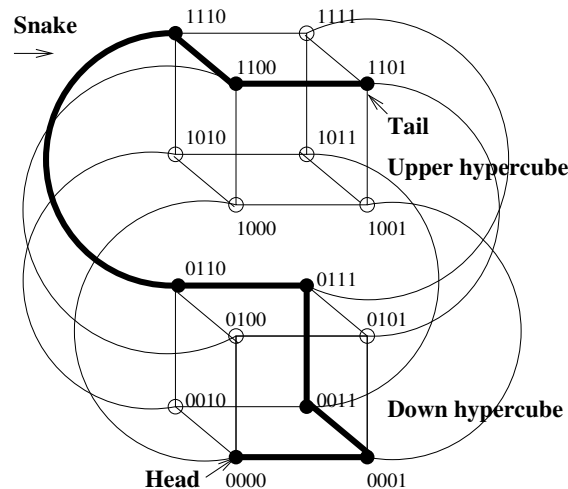


Figure 1: Hypercube of dimension 4 with a longest snake of length 7.

The *length* of a snake is the length of its open path, i.e., the numbers of edges that belong to the connected graph

Fitness Function	Random Generated - Seed(0)				Random Generated - Seed(1)			
	Minimum	Maximum	Average	σ	Minimum	Maximum	Average	σ
Eq.(1)	26	17,032	2,874.2	4,104.0	14	4,356.0	719.7	1,193.8
Eq.(2)	6	25,352	3,174.1	5,414.7	3	10,555.0	1,850.9	3,084.1
Eq.(3)	2	14,065	2,970.8	3,679.6	1	12,582.0	2,208.0	3,316.9
Eq.(4)	6	23,077	3,356.4	5,312.9	1	13,270.0	3,830.1	4,427.7
Eq.(5)	3	11,504	1,346.4	2,198.3	2	8,450.0	1,600.5	2,483.8
Eq.(6)	2	10,362	1,451.7	2,793.2	1	8,884.0	1,474.1	2,600.8
Eq.(7)	14	30,974	3,264.2	6,179.3	17	24,604.0	2,704.9	5,473.7

Table 1: Number of iterations of different fitness functions in finding longest snakes in a 4-dimensional hypercube. Initial Population Random Generated.

that represents it [8, 2]. The problem of finding longest snakes in hypercubes is a search problem in the 2^d -dimensional space, which is of order $O(2^{2^d})$, where d is the dimension of the hypercube. This makes the problem of finding longest snakes in hypercubes an appealing one for heuristic methods like genetic algorithms [4].

Seven Fitness Functions for Hunting Snakes. In order to evaluate the effect of the initial population in the develop of a GA, we are going to use the following fitness functions from [3], in order to find a longest snake in a hypercube of dimension 4.

Normalized Fitness Function(1):
$$F(I) = \left(\frac{\sum_{j=0}^{2^d-1} \sqrt{N_j - Penalty}}{\sum_{j=0}^{2^d-1} \sqrt{N_j}} \right) \left(\frac{Length(S)+1}{|S|} \right)$$

A Length Differential Fitness Function(2):
$$F(I) = \left(\frac{\sum_{j=0}^{2^d-1} \sqrt{N_j - Penalty}}{\sum_{j=0}^{2^d-1} \sqrt{N_j}} \right) * Length(S)$$

A Single Length Dependent Fitness Function(3):
$$F(I) = Length(S)$$

A Quadratic Fitness Function(4):
$$F(I) = (|S| - \#Lazy) * Length(S)$$

A Linear Fitness Function(5):
$$F(I) = (Length(S) - \#Lazy)$$

A Linear Fitness Function LazIs(6):
$$F(I) = (Length(S) - \#Lazy - \#Isolated)$$

A Rational Fitness Function(7):
$$F(I) = Length(S)/(1 + Penalty)$$

Effectiveness of Fitness Functions in Finding Longest Snakes in 4-Dimensional Hypercubes.

Evaluation of the Initial Population. The *initial population* is the seed for the algorithm. Good seeds should produce good results, or at least that is what we might expect¹. But besides that, how are those randomly generated chromosomes evaluated by the different fitness functions proposed in this abstract?

We performed four experiments: for experiments one and two, we initiate the initial populations randomly, with 10 individuals², using the java utility `java.util.Random(0)` and `java.util.Random(1)`. The results of the number of iterations performed by a GA in finding a longest snake in a hypercube of dimension 4 are in Table 1, using seven “different” fitness functions. We can observe that for seed(0) in average the best fitness functions were Functions Eq.5 and Eq.6, i.e., the linear ones that take into consideration lazy and isolated points. Now, when we use seed(1) all functions perform better for the maximum number of iterations and some in the average than when we use seed(0). Besides that, with seed(1), Function Eq.1 is now the best. What makes this better performance happened?

For seeing the difference of the fitness functions, we evaluate those using the t-test. We can look at Table 2 and see that Eq.5 and Eq.6 are not significant different—t-test equal to 0.881. But Eq.5—and somehow Eq.6—is significant different from Eq.1, Eq.3, and Eq.4. If we now observe the t-test when we use seed(1) Eq.5 continues being significant different from Eq.4 but not with Functions Eq.1 and Eq.3. Now, the function more significant different is Eq.1, which is significant different than Eq.2, Eq.3, Eq.4, and Eq.7. In both test, the functions that were more significant different than others were the ones, that in average, performed better.

If we now change the random generation of the first population and instead, generate half randomly and the other half using the information of the first half we obtain the following results. For the first case we generate the complement of the first half individuals, calling complement if an individual has 1 in a gene then it is replaced by 0 and so forth. For this case—See Table 3—we have that in average the best function was Eq.7 followed by Eq.1, and looking at Table 4 we can see that both are *not* significant different. Now Eq.7—and somehow Eq.1—is significant

¹It is known that, for the snake-in-the-box problem in higher dimensions, the algorithm can be seeded with a longest snake from the $(d-1)$ -hypercube [2] in order to improve results.

²We choose a small number to make the problem more interesting.

	<i>Random Generated - Seed(0)</i>						<i>Random Generated - Seed(1)</i>					
	Eq.(1)	Eq.(2)	Eq.(3)	Eq.(4)	Eq.(5)	Eq.(6)	Eq.(1)	Eq.(2)	Eq.(3)	Eq.(4)	Eq.(5)	Eq.(6)
Eq.(1)	-	-	-	-	-	-	-	-	-	-	-	-
Eq.(2)	0.811	-	-	-	-	-	0.08	-	-	-	-	-
Eq.(3)	0.911	0.852	-	-	-	-	0.02	0.69	-	-	-	-
Eq.(4)	0.672	0.893	0.688	-	-	-	0.00	0.08	0.13	-	-	-
Eq.(5)	0.084	0.115	0.050	0.061	-	-	0.11	0.69	0.49	0.04	-	-
Eq.(6)	0.118	0.121	0.076	0.111	0.881	-	0.15	0.64	0.38	0.01	0.86	-
Eq.(7)	0.781	0.948	0.825	0.948	0.131	0.170	0.06	0.42	0.69	0.4	0.28	0.31

Table 2: T-test results for iterations until longest possible found. Initial Population Random Generated

<i>Fitness Function</i>	<i>Pseudo Random Generated - Comp.</i>				<i>Pseudo Random Generated - Inverse</i>			
	<i>Minimum</i>	<i>Maximum</i>	<i>Average</i>	σ	<i>Minimum</i>	<i>Maximum</i>	<i>Average</i>	σ
Eq.(1)	4	6,738.0	1,861.7	1,970.5	2	9,662.0	2,019.7	2,568.3
Eq.(2)	5	11,566.0	1,983.5	3,079.7	3	16,571.0	2,236.4	3,681.2
Eq.(3)	5	16,940.0	3,012.8	4,048.2	3	20,861.0	3,407.9	4,905.9
Eq.(4)	16	18,507.0	2,576.4	3,854.9	4	9,377.0	1,985.8	2,864.8
Eq.(5)	12	31,143.0	4,027.1	6,048.9	2	11,442.0	1,604.7	2,940.5
Eq.(6)	6	16,278.0	2,457.2	4,109.0	1	11,342.0	1,313.9	2,227.6
Eq.(7)	2	10,553.0	1,800.1	2,550.2	3	18,221.0	1,954.9	4,186.5

Table 3: Number of iterations of different fitness functions in finding longest snakes in a 4-dimensional hypercube. Initial Population Pseudo-Random Generated.

different from Eq.5. And what happens to Eq.5 now? It has the worst performance in this test set—we should notice that Eq.5 was the best in average with initial population generated randomly with seed(0).

For the second case, we generate randomly the first half of the population and the other half taking the reverse order of each individual, i.e., if in the first half and individual is 0101010011110000, then the corresponding is 0000111100101010. In this case on average the best function was Eq.6 followed by Eq.5. Now the performance of each function was smoothed, i.e., there are not big gaps between them: for the minimum all where between 0 and 5, for the maximum all where between 9,000 and 21,000 and so forth. Function Eq.3 is the one that is more different that the others and is the one that has the worst performance—See Figure 2 where we can see that for initial population with seed(1) and with half of it inverted, the performance sum of all the average go down.

Conclusions & Future Work. We have built different fitness functions for solving the problem of hunting snakes in the box, and we use them with different initial population, in order to see the effect of the number of iterations. Each function has its strengths and drawbacks. The objective and the constraints are the two sides of the coin that are quite difficult to handle in this particular problem because we have tried to propose fitness functions with “natural” parameters on them. That is, our fitness functions try to use the information of the problem—the objective and constraints—or information that can be derived from the problem—the number of neighbors of a point—to parametrize the fitness functions. The performance of a genetic algorithm to find an approximate solution is systemic, in the sense that depends not only on the fitness function but also in the initial population, the number of individuals in the initial

	<i>Pseudo Random Generated - Complement</i>						<i>Pseudo Random Generated - Inverse</i>					
	Eq.(1)	Eq.(2)	Eq.(3)	Eq.(4)	Eq.(5)	Eq.(6)	Eq.(1)	Eq.(2)	Eq.(3)	Eq.(4)	Eq.(5)	Eq.(6)
Eq.(1)	-	-	-	-	-	-	-	-	-	-	-	-
Eq.(2)	0.86	-	-	-	-	-	0.81	-	-	-	-	-
Eq.(3)	0.21	0.32	-	-	-	-	0.14	0.36	-	-	-	-
Eq.(4)	0.28	0.56	0.66	-	-	-	0.96	0.75	0.2	-	-	-
Eq.(5)	0.08	0.11	0.46	0.24	-	-	0.56	0.44	0.06	0.53	-	-
Eq.(6)	0.48	0.65	0.62	0.89	0.27	-	0.29	0.24	0.05	0.33	0.69	-
Eq.(7)	0.92	0.83	0.17	0.34	0.09	0.46	0.94	0.76	0.19	0.97	0.73	0.44

Table 4: T-test results for iterations until longest possible found. Initial Population Pseudo Random Generated

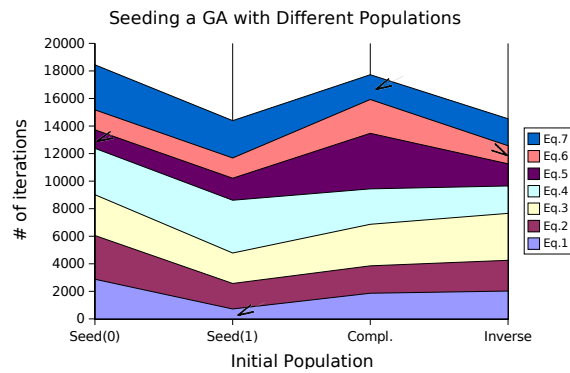


Figure 2: Different Performance of a GA with Different Fitness Functions and Initial Populations.

population, the crossover and mutation probabilities, the number of iterations, and so forth. However, it is possible that one of the strongest parameter is the fitness function in the sense that it is the one that is used in performing selection in looking for “good” solutions [3]. However, Those solutions are found with more or less computational resources, depending on the initial population. The initial population is the starting point in the search space, i.e., is the first step toward the hill. The performance of a genetic algorithm depends strongly on the initial population, and the random generation of it can be advantageous or disadvantageous depending of the problem to be solved [1]. A good knowledge of the problem to be solved, can help in the generation of the initial population, but sometimes the complete understanding of proper regions in the search space to look for, could be difficult to visualize [1].

Our work is on going, scaling to higher hypercubes, reporting our results, and trying to improve our algorithm with new parameters that accommodate smoothly to the problem itself, showing new ways to generate GA parameters.

References

- [1] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, 1996.
- [2] D. A. Casella and W. D. Potter. New lower bounds for the snake-in-the-box problem: Using evolutionary techniques to hunt for snakes. In *Proceedings of American Association for Artificial Intelligence*, pages 264–268, 2004.
- [3] Pedro Diaz-Gomez and Dean Hougen. *Genetic Algorithms for Hunting Snakes in Hypercubes: Fitness Function Analysis and Open Questions*, 2006.
- [4] W.D Potter et al. Using the genetic algorithm to find snake-in-the-box codes. In *7-th International Conference On Industrial & Engineering Applications Of Artificial Intelligence and expert Systems*, pages 421–426, 1994.
- [5] David S. Greenberg and Sandeep N. Bhatt. Routing multiple paths in hypercubes. In *Proceedings of the second annual ACM symposium on Parallel algorithms and architectures*, pages 45–54, 1990.
- [6] John Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1992.
- [7] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1998.
- [8] Dayanand S. Rajan. Maximal and reversible snakes in hypercubes. Computer Science and Physics Department, Roanoke College, USA. Accessed January 2006.
- [9] Lakshmiarahan S. and Dhall Sudarshan. *Analysis and design of parallel algorithms:arithmetic and matrix problem*. MacGraw-Hill, 1990.